Adversarial Feature Matching for Text Generation

Yizhe Zhang [1] Zhe Gan [1] Kai Fan [1] Zhi Chen [1] Ricardo Henao [1] Dinghan Shen [1] Lawrence Carin [1]

Abstract

The Generative Adversarial Network (GAN) has achieved great success in generating realistic (real-valued) synthetic data. However, convergence issues and difficulties dealing with discrete data hinder the applicability of GAN to text. We propose a framework for generating realistic text via adversarial training. We employ a long short-term memory network as generator, and a convolutional network as discriminator. Instead of using the standard objective of GAN, we propose matching the high-dimensional latent feature distributions of real and synthetic sentences, via a kernelized discrepancy metric. This eases adversarial training by alleviating the mode-collapsing problem. Our experiments show superior performance in quantitative evaluation, and demonstrate that our model can generate realistic-looking sentences.

1. Introduction

Generating meaningful and coherent sentences is central to many natural language processing applications. The general idea is to estimate a distribution over sentences from a corpus, then use it to sample realistic-looking sentences. This task is important because it enables generation of *novel* sentences that preserve the semantic and syntactic properties of real-world sentences, while being potentially different from any of the examples used to estimate the model. For instance, in the context of dialog generation, it is desirable to generate answers that are more diverse and less generic (Li et al., 2016).

One simple approach consists of first learning a latent space to represent (fixed-length) sentences using an encoder-decoder (autoencoder) framework based on Recurrent Neural Networks (RNNs) (Cho et al., 2014; Sutskever et al., 2014), then generate synthetic sentences by decoding random samples from this latent space. However, this approach often fails to generate realistic sentences from arbitrary latent representations. The reason for this is that, when mapping sentences to their latent representations using an autoencoder, the mappings usually cover a small but structured region of the latent space, which corresponds to a manifold embedding (Bowman et al., 2016). In practice, most regions of the latent space do not necessarily map (decode) to realistic sentences. Consequently, randomly sampling latent representations often yields nonsensical sentences. Recent work by Bowman et al. (2016) has attempted to generate more diverse sentences via RNN-based variational autoencoders. However, they did not address the fundamental problem that the posterior distribution over latent variables does not appropriately cover the latent space.

Another underlying challenge of generating realistic text relates to the nature of the RNN. During inference, the RNN generates words in sequence from previously *generated* words, contrary to learning, where ground-truth words are used every time. As a result, error accumulates proportional to the length of the sequence, *i.e.*, the first few words look reasonable, however, quality deteriorates quickly as the sentence progresses. Bengio et al. (2015) coined this phenomenon *exposure bias*. Toward addressing this problem, Bengio et al. (2015) proposed the scheduled sampling approach. However, Huszár (2015) showed that scheduled sampling is a fundamentally inconsistent training strategy, in that it produces largely unstable results in practice.

The Generative Adversarial Network (GAN) (Goodfellow et al., 2014) is an appealing and natural answer to the above issues. GAN matches the distributions of synthetic and real data by introducing an adversarial game between a *generator* and a *discriminator*. The GAN objective seeks to constitute a generator, that functionally maps samples from a given (simple) prior distribution, to synthetic data that appear to be *realistic*. The GAN setup explicitly seeks that the latent representations from real data (via encoding) be distributed in a manner consistent with the specified prior (*e.g.*, Gaussian or uniform). Due to the nature of adversarial training, the discriminator compares real and synthetic sentences, rather than their individual words, which in principle should alleviate the exposure-bias issue. Recent work (Lamb et al., 2016) has incorporated an additional discriminator to train a sequence-to-sequence language model that better preserves

[1]Duke University, Durham, NC, 27708. Correspondence to: Yizhe Zhang <yizhe.zhang@duke.edu>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017. Copyright 2017 by the author(s).

long-term dependencies.

Effort has also been made to generate realistic-looking sentences via adversarial training. For instance, by borrowing ideas from reinforcement learning, Yu et al. (2017); Li et al. (2017) treat the sentence generation as a sequential decision making process. Despite the success of these methods, two fundamental problems of the GAN framework limit their use in practice: (*i*) the generator tends to produce a single observation for multiple latent representations, *i.e.*, mode collapsing (Metz et al., 2017), and (*ii*) the generator's contribution to the *learning signal* is insubstantial when the discriminator is close to its local optimum, *i.e.*, vanishing gradient behavior (Arjovsky & Bottou, 2017).

In this paper we propose a new framework, *TextGAN*, to alleviate the problems associated with generating realistic-looking sentences via GAN. Specifically, the Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997) RNN is used as generator, and the Convolutional Neural Network (CNN) (Kim, 2014) is used as discriminator. We consider a kernel-based moment-matching scheme over a Reproducing Kernel Hilbert Space (RKHS), to force the empirical distributions of real and synthetic sentences to have matched moments in latent-feature space. As a consequence, our approach ameliorates the mode-collapsing issue associated with standard GAN training. This strategy encourages the model to learn representations that are both informative of the original sentences (via the autoencoder) and discriminative w.r.t. synthetic sentences (via the discriminator). We also propose several complementary techniques, including initialization strategies and discretization approximations to ease GAN training, and to achieve superior performance compared to related approaches.

2. Model

2.1. Generative Adversarial Networks

GAN (Goodfellow et al., 2014) aims to obtain the equilibrium of the following optimization objective

$$\mathcal{L}_{GAN} = \mathbb{E}_{x\sim p_x} \log D(x) + \mathbb{E}_{z\sim p_z} \log[1 - D(G(z))], \quad (1)$$

where $\mathcal{L}_{GAN}$ is maximized w.r.t. $D(\cdot)$ and minimized w.r.t. $G(\cdot)$. Note that the first term of (1) does not depend on $G(\cdot)$. Observed (real) data, x, are sampled from empirical distribution $p_x(\cdot)$. The latent code, z, that feeds into the generator, $G(z)$, is drawn from a simple prior distribution $p_z(\cdot)$. When the discriminator is *optimal*, solving this adversarial game is equivalent to minimizing the Jenson-Shannon Divergence (JSD) (Arjovsky & Bottou, 2017) between the real data distribution $p_x(\cdot)$ and the synthetic data distribution $p_{\tilde{x}}(\cdot) \triangleq p(G(z))$, where $z \sim p_z(\cdot)$ (Goodfellow et al., 2014). However, in most cases, the saddle-point solution of the objective in (1) is intractable. Therefore, a procedure to iteratively update $D(\cdot)$ and $G(\cdot)$ is often applied.

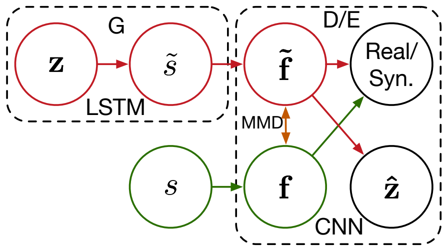

Figure 1. Model scheme of TextGAN. Latent codes z are fed through a generator $G(\cdot)$, to produce synthetic sentence $\tilde{s}$. Synthetic and real sentences ($\tilde{s}$ and s) are fed into a binary discriminator $D(\cdot)$, for real *vs.* fake (synthetic) prediction, and also for latent code reconstruction $\hat{z}$. $\tilde{\boldsymbol{f}}$ and $\boldsymbol{f}$ represent features of $\tilde{s}$ and s, respectively.

Arjovsky & Bottou (2017) pointed out that the standard GAN objective in (1) suffers from an unstably weak learning signal when the discriminator gets close to local optimal, due to the gradient-vanishing effect. This is because the JSD implied by the original GAN loss becomes a constant if $p_x(\cdot)$ and $p_{\tilde{x}}(\cdot)$ share no support, thus minimizing the JSD yields no learning signal. This problem also exists in the recently proposed energy-based GAN (EBGAN) (Zhao et al., 2017), as the distance metric implied by EBGAN is the Total Variance Distance (TVD), which has the same issue w.r.t. JSD, as shown by Arjovsky et al. (2017).

2.2. TextGAN

Given a sentence corpus $\mathcal{S}$, instead of directly optimizing the objective from standard GAN in (1), we adopt an approach that is similar to the feature matching scheme of Salimans et al. (2016). Specifically, we consider the objective

$$\mathcal{L}_D = \mathcal{L}_{GAN} - \lambda_r \mathcal{L}_{recon} + \lambda_m \mathcal{L}_{MMD^2} \quad (2)$$

$$\mathcal{L}_G = \mathcal{L}_{MMD^2} \quad (3)$$

$$\mathcal{L}_{GAN} = \mathbb{E}_{s\sim\mathcal{S}} \log D(s) + \mathbb{E}_{z\sim p_z} \log[1 - D(G(z))]$$

$$\mathcal{L}_{recon} = ||\hat{z} - z||^2 ,$$

where $\mathcal{L}_D$ and $\mathcal{L}_G$ are iteratively maximized w.r.t $D(\cdot)$ and minimized w.r.t. $G(\cdot)$, respectively. $\mathcal{L}_{GAN}$ is the standard objective of GAN in (1). $\mathcal{L}_{recon}$ is the Euclidean distance between the reconstructed latent code, $\hat{z}$, and the original code, z, drawn from prior distribution $p_z(\cdot)$. We denote the synthetic sentences as $\tilde{s} \triangleq G(z)$, where $z \sim p_z(\cdot)$. $\mathcal{L}_{MMD^2}$ represents the Maximum Mean Discrepancy (MMD) (Gretton et al., 2012) between the empirical distribution of sentence embeddings $\tilde{\boldsymbol{f}}$ and $\boldsymbol{f}$, for synthetic and real data, respectively. The model framework is illustrated in Figure 1 and detailed below.

We first consider $\mathcal{L}_G$ in (3). The generator $G(\cdot)$ attempts to adjust itself to produce synthetic sentence $\tilde{s}$, with features

$\tilde{\boldsymbol{f}}$, encoded by $D(\cdot)$, to mimic the real sentence features $\boldsymbol{f}$ (also encoded by $D(\cdot)$). This is achieved by matching the empirical distributions of $\tilde{\boldsymbol{f}}$ and $\boldsymbol{f}$ via the MMD objective.

Concisely, MMD measures the mean squared difference between two sets of samples $\mathcal{X}$ and $\mathcal{Y}$, where $\mathcal{X} = \{x_i\}_{i=1:N_x}$, $x_i \in \mathbb{R}^d$, $\mathcal{Y} = \{y_i\}_{i=1:N_y}$, $y_i \in \mathbb{R}^d$, d is the dimensionality of the samples, and N_x and N_y are sample sizes for $\mathcal{X}$ and $\mathcal{Y}$, respectively. The MMD metric characterizes the differences between $\mathcal{X}$ and $\mathcal{Y}$ over a Reproducing Kernel Hilbert Space (RKHS), $\mathcal{H}$, associated with kernel function $k(\cdot) : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$. The kernel can be written as an inner product over $\mathcal{H}$: $k(x, x') = \langle k(x, \cdot), k(x', \cdot)\rangle_{\mathcal{H}}$, and $\boldsymbol{\phi}(x) \triangleq k(x, \cdot) \in \mathcal{H}$ is denoted as the *feature mapping* (Gretton et al., 2012). Formally, the MMD for two empirical distributions $\mathcal{X}$ and $\mathcal{Y}$ is given by

$$\begin{aligned}\mathcal{L}_{MMD^2} &= ||\mathbb{E}_{x\sim\mathcal{X}}\boldsymbol{\phi}(x) - \mathbb{E}_{y\sim\mathcal{Y}}\boldsymbol{\phi}(y)||^2_{\mathcal{H}} \\ &= \mathbb{E}_{x\sim\mathcal{X}}\mathbb{E}_{x'\sim\mathcal{X}}[k(x, x')] \\ &+ \mathbb{E}_{y\sim\mathcal{Y}}\mathbb{E}_{y'\sim\mathcal{Y}}[k(y, y')] - 2\mathbb{E}_{x\sim\mathcal{X}}\mathbb{E}_{y\sim\mathcal{Y}}[k(x, y)].\end{aligned} \quad (4)$$

Note that $\mathcal{L}_{MMD^2}$ reaches its minimum when the two empirical distributions $\mathcal{X}$ and $\mathcal{Y}$ (in general) match exactly. For example, with a polynomial kernel, $k(x, y) = (x^T y + c)^L$, minimizing $\mathcal{L}_{MMD^2}$ can be understood as matching moments of two empirical distributions up to order L. With a universal kernel like the Gaussian kernel, $k(x, y) = \exp(-\frac{||x-y||^2}{2\sigma})$, with bandwidth σ, minimizing the MMD objective will match moments of all orders (Gretton et al., 2012). Here, we use MMD to match the empirical distribution of $\tilde{\boldsymbol{f}}$ and $\boldsymbol{f}$ using a Gaussian kernel.

The adversarial discriminator $D(\cdot)$ associated with the loss in (2) aims to produce sentence features that are most *discriminative*, *representative* and *challenging*. These aims are explicitly represented as the three components of (2), namely, (*i*) $\mathcal{L}_{GAN}$ requires $\tilde{\boldsymbol{f}}$ and $\boldsymbol{f}$ to be discriminative of real and synthesized sentences; (*ii*) $\mathcal{L}_{recon}$ requires $\tilde{\boldsymbol{f}}$ and $\boldsymbol{f}$ to preserve maximum reconstruction information for the latent code z that generates synthetic sentences; and (*iii*) $\mathcal{L}_{MMD^2}$ forces $D(\cdot)$ to select the most challenging features for the generator to match.

In the situation for which simple features are enough for the discrimination/reconstruction task, this additional loss seeks to estimate complex features that are difficult for the current generator, thus improving in terms of generation ability. In our experience, we find the reconstruction and MMD loss in D serve as regularizer to the binary classification loss, in that by adding these losses, discriminator features tend to be more spread-out in the feature space.

In summary, the adversarial game associated with (2) and (3) is the following: $D(\cdot)$ attempts to select informative sentence features, while $G(\cdot)$ aims to match these features. Parameters λ_r and λ_m act as trade-off between discrimination ability, and reconstruction and moment matching precision, respectively. We argue that this framework has several advantages over the standard GAN objective in (1).

The original GAN objective has been shown to be prone to mode collapsing, especially when the so-called $\log D$ alternative for the generator loss is used (Metz et al., 2017), *i.e.*, replacing the second term of (1) by $-\mathbb{E}_{z\sim p_z} \log[D(G(z))]$. This is because when $\log D$ is used, fake-looking samples are penalized more severely than less diverse samples (Arjovsky & Bottou, 2017), thus grossly underestimating the variance of latent features. The loss in (3), on the other hand, forces the generator to produce highly diverse sentences to match the variation of real sentences, by latent moment matching, thus alleviating the mode-collapsing problem. We believe that leveraging MMD is general enough to be useful as a framework in other data domains, *e.g.*, images. Presumably, the discrete nature of text data makes standard GAN prone to mode-collapsing. This is manifested by close neighbors in latent code space producing the same text output. In our approach, MMD and feature matching are introduced to alleviate mode collapsing with text data as motivating domain. However, whether such an objective is free from the convergence issues of the standard GAN, due to vanishing gradient from the generator, is known to be problem specific (Arjovsky & Bottou, 2017).

Arjovsky & Bottou (2017) demonstrated that JSD yields weak gradient signals when the real and synthetic data are far apart. To deliver stable gradients, a smoother distance metric over the data domain is required. In (4), we are essentially employing a Neural Network (NN) embedding via Gaussian kernel for matching s and $\tilde{s}$, *i.e.*, $k_s(s, s') = \boldsymbol{\phi}(g(s))^T \boldsymbol{\phi}(g(s'))$, where $g(\cdot)$ denotes the NN embedding that maps from the data to the feature domain. Under the assumption that $g(\cdot)$ is a bijective mapping, *i.e.*, distinct sentences have different embedded feature vectors, in the Supplementary Material we prove that if the original kernel function $k(x, y) = \boldsymbol{\phi}(x)^T \boldsymbol{\phi}(y)$ is universal, the composed kernel $k_s(s, s')$ is also universal. As shown in Gretton et al. (2012), the MMD is a proper metric when the kernel is universal. In fact, if the kernel function is universal, the MMD metric will be no worse than TVD in terms of vanishing gradients (Arjovsky et al., 2017). However, if the bandwidth of the kernel is too small, much smaller than the average distance between data points, the vanishing gradient problem remains (Arjovsky et al., 2017).

Additionally, seeking to match the sentence features provides a more achievable and informative objective than directly trying to mislead the discriminator as in standard GAN. Specifically, the loss in (3) implies a clearer aim for the generator, as it requires matching the latent features (distribution-wise) as opposed to uniquely trying to fake a binary classifier.

Note that if the latent features from real and synthetic data have similar distributions it is unlikely that the discriminator, that uses these features as inputs, will be able to tell them apart. Implementation-wise, the updating signal from the generator does not need to propagate all the way back from the discriminator, but rather directly from the features layer, thus less prone to fading. We believe there may be other possible approaches for text generation using GAN, however, we hope to provide a first attempt toward overcoming some of the difficulties associated with it.

2.3. Alternative (data efficient) objectives

One limitation of the proposed approach is that the dimensionality of features $\tilde{\boldsymbol{f}}$ and $\boldsymbol{f}$ could be much larger than the size of the subset of data (minibatch) used during learning, hence the empirical distribution may not be sufficiently representative. In fact, a reliable Gaussian kernel MMD two-sample test generally requires the size of the minibatch to be proportional to the number of dimensions (Ramdas et al., 2014). To alleviate this issue, we consider two strategies.

Compressing network We map $\tilde{\boldsymbol{f}}$ and $\boldsymbol{f}$ into a lower-dimensional feature space using a *compressing network* with fully connected layers, also learned by $D(\cdot)$. This is sensible because the discriminator will still encourage the most challenging features to be abstracted (compressed) from the original features $\tilde{\boldsymbol{f}}$ and $\boldsymbol{f}$. This approach provides significant computational savings, as computation of the MMD in (4) scales with $\mathcal{O}(d^2 d_f)$, where d_f denotes the dimensionality of the feature vector. However, a lower-dimensional mapping may miss valuable information. Besides, finding the optimal mapping dimension may be difficult in practice. There exists a tradeoff between fast estimation and a richer feature vector, by setting d_f appropriately.

Gaussian covariance matching We could also avoid using the *kernel trick*, as was used in (4). Instead, we can replace $\mathcal{L}_{MMD^2}$ by $\mathcal{L}_G^{(c)}$ (below), where we accumulate (Gaussian) sufficient statistics from multiple minibatches, thus alleviating the inadequate-minibatch-size issue. Specifically,

$$\mathcal{L}_G^{(c)} = \mathrm{tr}(\tilde{\mathbf{\Sigma}}^{-1}\mathbf{\Sigma} + \mathbf{\Sigma}^{-1}\tilde{\mathbf{\Sigma}}) + (\tilde{\boldsymbol{\mu}} - \boldsymbol{\mu})^T(\tilde{\mathbf{\Sigma}}^{-1} + \mathbf{\Sigma}^{-1})(\tilde{\boldsymbol{\mu}} - \boldsymbol{\mu}) \, , \qquad (5)$$

where $\tilde{\mathbf{\Sigma}}$ and $\mathbf{\Sigma}$ represent the covariance matrices of synthetic and real sentence feature vectors $\tilde{\boldsymbol{f}}$ and $\boldsymbol{f}$, respectively. $\tilde{\boldsymbol{\mu}}$ and $\boldsymbol{\mu}$ denote the mean vectors of $\tilde{\boldsymbol{f}}$ and $\boldsymbol{f}$, respectively. By setting $\tilde{\mathbf{\Sigma}} = \mathbf{\Sigma} = \mathbf{I}$, (5) reduces to the first-moment feature matching technique from Salimans et al. (2016). Note that this loss $\mathcal{L}_G^{(c)}$ is an upper bound of the JSD (omitting constant, proved in the Supplementary Material) between two multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \mathbf{\Sigma})$ and $\mathcal{N}(\tilde{\boldsymbol{\mu}}, \tilde{\mathbf{\Sigma}})$, which is more tractable than directly minimizing JSD. The feature vectors used in (5) are the neural net outputs before applying any non-linear activation function. We note that the Gaussian assumption may still be strong in many cases. In practice, we use a moving average of the most recent m minibatches for estimating all sufficient statistics $\tilde{\mathbf{\Sigma}}, \mathbf{\Sigma}, \tilde{\boldsymbol{\mu}}$ and $\boldsymbol{\mu}$. Further, $\tilde{\mathbf{\Sigma}}$ and $\mathbf{\Sigma}$ are initialized to be $\mathbf{I}$ to prevent numerical problems.

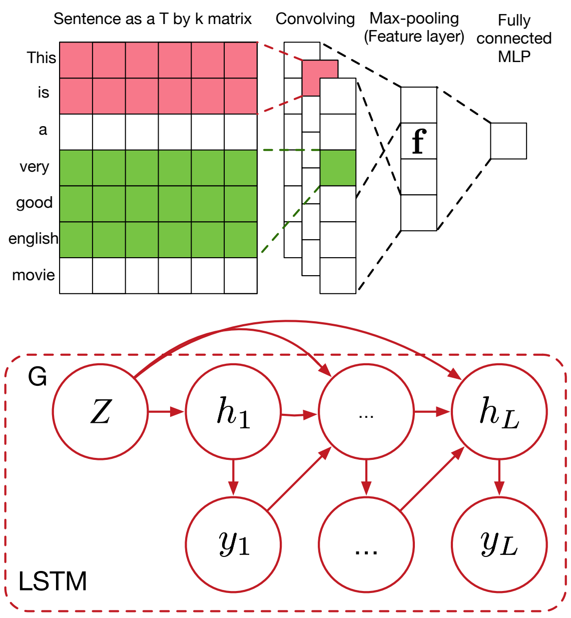

Figure 2. Top: CNN-based sentence discriminator/encoder. Bottom: LSTM sentence generator.

2.4. Model specification

Let w^t denote the t-th word in sentence s. Each word w^t is embedded into a k-dimensional word vector $\boldsymbol{x}_t = \mathbf{W}_e[w^t]$, where $\mathbf{W}_e \in \mathbb{R}^{k \times V}$ is a (learned) word embedding matrix, V is the vocabulary size, and notation $\mathbf{W}_e[v]$ denotes the v-th column of matrix $\mathbf{W}_e$.

CNN discriminator We use the CNN architecture in Kim (2014); Collobert et al. (2011) for sentence encoding. It consists of a convolution layer and a max-pooling operation over the entire sentence for each feature map. A sentence of length T (padded where necessary) is represented as a matrix $\mathbf{X} \in \mathbb{R}^{k \times T}$, by concatenating its word embeddings as columns, *i.e.*, the t-th column of $\mathbf{X}$ is $\boldsymbol{x}_t$.

As shown in Figure 2(top), a convolution operation involves a filter $\mathbf{W}_c \in \mathbb{R}^{k \times h}$, applied to a window of h words to produce a new feature. Following Collobert et al. (2011), we induce a latent feature map $\boldsymbol{c} = \gamma(\mathbf{X} * \mathbf{W}_c + \boldsymbol{b}) \in \mathbb{R}^{T-h+1}$, where $\gamma(\cdot)$ is a nonlinear activation function (we use the hyperbolic tangent, tanh), $\boldsymbol{b} \in \mathbb{R}^{T-h+1}$ is a bias vector, and $*$ denotes the convolutional operator. We then apply a max-over-time pooling operation (Collobert et al., 2011) to the feature map and take its maximum value, *i.e.*, $\hat{c} = \max\{\boldsymbol{c}\}$, as the feature corresponding to this particular filter. Convolving the same filter with the h-gram at every position in the sentence allows features to be extracted independently of their position in the sentence. This pooling scheme tries to capture the most salient feature, *i.e.*, the one with the highest value, for each feature map, effectively filtering out less informative compositions of words. Further, this pooling scheme also guarantees that the extracted features are independent of the length of the input sentence.

The above process describes how one feature is extracted from one filter. In practice, the model uses multiple filters with varying window sizes. Each filter can be considered as a *linguistic feature* detector that learns to recognize a specific class of h-grams. Assume we have m window sizes, and for each window size, we use p filters, then we obtain a mp-dimensional vector $\boldsymbol{f}$ to represent a sentence. On top of this mp-dimensional feature vector, we specify a softmax layer to map the input sentence to an output $D(\mathbf{X}) \in [0, 1]$, representing the probability of $\mathbf{X}$ being from the data distribution (real), rather than from the adversarial generator (synthesized).

There are other CNN architectures in the literature (Kalchbrenner et al., 2014; Hu et al., 2014; Johnson & Zhang, 2015). We adopt the CNN model of Kim (2014); Collobert et al. (2011) due to its simplicity and excellent performance on sentence classification tasks.

LSTM generator We specify an LSTM generator to translate a latent code vector, $\boldsymbol{z}$, into a synthetic sentence $\tilde{s}$. This is illustrated in Figure 2(bottom). The probability of a length-T sentence, $\tilde{s}$, given the encoded feature vector, $\boldsymbol{z}$, is defined as

$$p(\tilde{s}|\boldsymbol{z}) = p(\tilde{w}^1|\boldsymbol{z}) \prod_{t=2}^{T} p(\tilde{w}^t|\tilde{w}^{<t}, \boldsymbol{z}) \,, \tag{6}$$

where $\tilde{w}^t$ denotes the t-th generated token. Specifically, we generate the first word $\tilde{w}^1$, deterministically from $\boldsymbol{z}$, with $(\tilde{w}^1|\boldsymbol{z}) = \text{argmax}(\mathbf{V}\boldsymbol{h}_1)$, where $\boldsymbol{h}_1 = \tanh(\mathbf{C}\boldsymbol{z})$. Bias terms are omitted for simplicity. All other words in the sentence are sequentially generated using the RNN, based on previously generated words, until the end-sentence symbol is generated. The t-th word $\tilde{w}^t$ is generated as $(\tilde{w}^t|\tilde{w}^{<t}, \boldsymbol{z}) = \text{argmax}(\mathbf{V}\boldsymbol{h}_t)$, where $< t \triangleq \{1, \ldots, t-1\}$, and the hidden units $\boldsymbol{h}_t$ are recursively updated through $\boldsymbol{h}_t = \mathcal{U}(\boldsymbol{y}_{t-1}, \boldsymbol{h}_{t-1}, \boldsymbol{z})$. $\mathbf{V}$ is a weight matrix used for computing a distribution over words. The input $\boldsymbol{y}_{t-1}$ for the t-th step is the embedding vector of the previous generated word $\tilde{w}^{t-1}$, *i.e.*,

$$\boldsymbol{y}_{t-1} = \mathbf{W}_e[\tilde{w}^{t-1}] \,. \tag{7}$$

The synthetic sentence $\tilde{s} = [\tilde{w}^1, \cdots, \tilde{w}^L]$ is deterministically obtained given $\boldsymbol{z}$ by concatenating the generated words. In experiments, the transition function, $\mathcal{U}(\cdot)$, is implemented with an LSTM (Hochreiter & Schmidhuber, 1997). Details are provided in the Supplementary Material.

2.5. Training Techniques

Soft-argmax approximation To train the generator $G(\cdot)$, which contains discrete variables, direct application of the gradient estimation may be difficult (Yu et al., 2017). Score-function-based approaches, such as the REINFORCE algorithm (Williams, 1992), achieve unbiased gradient estimation for discrete variables using Monte Carlo estimation. However, in our experiments, we found that the variance of the gradient estimation is very large, which is consistent with Maddison et al. (2017). Here we consider a *soft-argmax* operator (Zhang et al., 2016), similar to the Gumbel-softmax (Gumbel & Lieblein, 1954; Jang et al., 2017), when performing learning, as an approximation to (7):

$$\boldsymbol{y}_{t-1} = \mathbf{W}_e \text{softmax}(\mathbf{V}\boldsymbol{h}_{t-1} \odot L) \,. \tag{8}$$

where $\odot$ represents the element-wise product. Note that when $L \to \infty$, this approximation approaches (7).

Pre-training Previous literature (Goodfellow et al., 2014; Salimans et al., 2016) has discussed the fundamental difficulty of training GANs using gradient-based methods. In general, gradient descent optimization schemes may fail to converge to the equilibrium by moving along the orbit trajectory among saddle points (Salimans et al., 2016). Intuitively, good initialization can facilitate convergence. Toward this end, we initialize the LSTM parameters of the generator by pre-training a standard CNN-LSTM autoencoder (Gan et al., 2016). For the discriminator/encoder initialization, we use a *permutation training* strategy. For each sentence in the corpus, we randomly swap two words to construct a slightly *tweaked* sentence counterpart. The discriminator is pre-trained to distinguish the tweaked sentences from the true sentences. The swapping operation is preferred here because it constitutes a much more challenging task for the discriminator to learn, compared to adding or deleting words, where the structure of real sentences is more strongly disrupted, thus making it easier for the discriminator. The permutation pre-training is important because it requires the discriminator to learn features characteristic of sentences' long dependencies. We empirically found this provides a better initialization (compared to no pre-training) for the discriminator to learn good features.

We also utilized other training techniques to stabilize training, such as soft-labeling (Salimans et al., 2016). Details of these are provided in the Supplementary Material.

3. Related Work

Generative Moment Matching Networks (GMMNs) (Dziugaite et al., 2015; Li et al., 2015) are closely related to our approach. However, these methods either directly match the empirical distribution in the data domain, or extract features using a pre-trained autoencoder (Li et al., 2015). If the goal is to perform matching in the data domain when generating sentences, the dimensionality of input data would be $T \times k$ (higher than 10,000 in our case). Note that the minibatch size required to obtain reasonable statistical power grows linearly with the number of dimension (Ramdas et al., 2014), and the computational cost of MMD grows quadratically with the size of data points. Therefore, directly applying GMMNs is often computationally prohibitive. Furthermore, directly matching in the data domain via GMMNs implies word-by-word discrepancy, which yields less smooth gradients. This happens because a word-by-word discrepancy ignores sentence structure. For example, two sentences "a boy is swimming" and "boy is swimming" will be far apart in a word-by-word metric, when they are indeed close in a sentence-by-sentence feature space.

A two-step method, where a feature encoder is generated first as in Li et al. (2015) helps alleviate the problems above. However, in Li et al. (2015) the feature encoder is fixed once pre-trained, limiting the potential to adjust features during the training phase. Alternatively, our approach matches the real and synthetic data on a sentence feature space, where features are *dynamically* and *adversarially* adapted to focus on the most challenging features for the generator to mimic. In addition, features are designed to maintain both discrimination and reconstruction ability, instead of merely focusing on reconstruction as in Li et al. (2015).

Recent work considered combining autoencoders or variational autoencoders (Kingma & Welling, 2014) with GAN (Zhao et al., 2017; Larsen et al., 2016; Makhzani et al., 2015; Mescheder et al., 2017; Wang & Liu, 2016). They demonstrated superior performance on image generation. Our approach is similar to these approaches; however, we attempt to learn the reconstruction of the latent code, instead of the input data (sentences). Donahue et al. (2017); **?** learned a reverse mapping from data space to latent space. In our approach we enforce the discriminator and encoder to share a latent structure, with the aim of learning a representation for both discrimination and latent code reconstruction. Chen et al. (2016) maximized the mutual information between the generated data and the latent codes by leveraging a network-adapted variational proposal distribution. In our case, we minimize the distance between the original and reconstructed latent codes.

Our approach attempts to minimize a NN-based embedded MMD distance of two empirical distributions. Aside from MMD, kernel-based discrepancy metrics such as kernelized Stein discrepancy (Liu et al., 2016; Wang & Liu, 2016) have been shown to be computationally tractable, while maintaining statistical power. We leave the investigation of using Stein for moment matching as a promising future direction. Wasserstein GAN (Arjovsky et al., 2017) considers an Earth-Mover (EM) distance of the real data and synthetic data distribution, instead of the JSD as in standard GAN (Goodfellow et al., 2014) or TVD as in Zhao et al. (2017). The EM metric yields stable gradients, thus avoiding the collapsing mode and vanishing gradient problem of the latter two. We note that our approach is equivalent to minimizing a MMD loss over the data domain, however, with a NN-based embedded Gaussian kernel. As shown in Arjovsky et al. (2017), MMD is a proper metric when the kernel is universal. Because of the similarity of the conditions, our approach enjoys the advantages of Wasserstein GAN, namely, ameliorating the gradient vanishing problems.

4. Experiments

Data and Experimental Setup Our model is trained using a combination of two datasets: (*i*) the BookCorpus dataset (Zhu et al., 2015), which consists of 70 million sentences from over 7000 books; and (*ii*) the ArXiv dataset, which consists of 5 million sentences from abstracts of papers from various subjects, obtained from the arXiv website. The motivation for merging two different corpora is to investigate whether the model can generate sentences that integrate both scientific and informal writing styles. We randomly choose 0.5 million sentences from BookCorpus and 0.5 million sentences from arXiv to construct training and validation sets, *i.e.*, 1 million sentences for each. For testing, we randomly select 25,000 sentences from both corpus, for a total of 50,000 sentences.

We train the generator and discriminator/encoder iteratively. Provided that the LSTM generator typically involves more parameters and is more difficult to train than the CNN discriminator, we perform one optimization step for the discriminator for every $K = 5$ steps of the generator. We use a mixture of 5 isotropic Gaussian (RBF) kernels with different bandwidths σ as in Li et al. (2015). Bandwidth parameters are selected to be close to the median distance (in our case around 20) of feature vectors encoded from real sentences. λ_r and λ_m are selected based on the performance on the validation set. The validation performance is evaluated by loss of generator and corpus-level BLEU score (Papineni et al., 2002), described below.

For the CNN discriminator/encoder, we use filter windows (h) of sizes {3,4,5} with 300 feature maps each, hence each sentence is represented as a 900-dimensional vector. The dimensionality of $\boldsymbol{z}$ and $\hat{\boldsymbol{z}}$ is also 900. The feature vector is then fed into a 900-200-2 fully connected network for the discriminator and 900-900-900 for encoder, with sigmoid

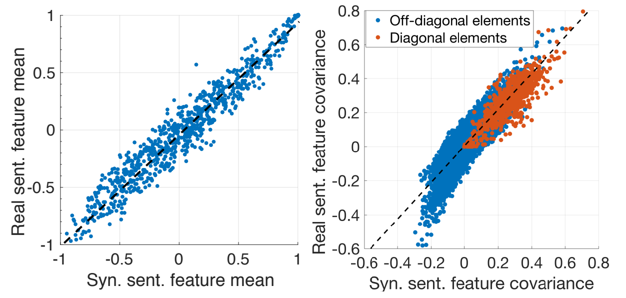

Figure 3. Moment matching comparison. Left: expectations of latent features from real *vs.* synthetic data. Right: elements of $\tilde{\mathbf{\Sigma}}_{i,j,\boldsymbol{f}}$ *vs.* $\tilde{\mathbf{\Sigma}}_{i,j,\tilde{\boldsymbol{f}}}$, for real and synthetic data, respectively.

activation units connecting the intermediate layers and softmax/tanh units for the top layer of discriminator/encoder. We did not observe performance changes by adding dropout. For the LSTM sentence generator, we use one hidden layer of 500 units.

Gradients are clipped if the norm of the parameter vector exceeds 5 (Sutskever et al., 2014). Adam (Kingma & Ba, 2015) with learning rate 5×10^{-5} for both discriminator and generator is utilized for optimization. The size of the minibatch is set to 256.

Both the generator and the discriminator are pre-trained using the strategies described in Section 2. We also employed a *warm-up* training during the first two epochs, as we found it improves convergence during the initial stage of learning. Specifically, we use a mean-matching objective for the generator loss, *i.e.*, $||\mathbb{E}\boldsymbol{f} - \mathbb{E}\tilde{\boldsymbol{f}}||^2$, as in Salimans et al. (2016). Further details of the experimental design are provided in the the Supplementary Material. All experiments are implemented in Theano (Bastien et al., 2012), using one NVIDIA GeForce GTX TITAN X GPU with 12GB memory. The model was trained for 50 epochs in roughly 3 days. Learning curves are shown in the Supplementary Material.

Matching feature distributions We first examine the generator's ability to produce synthetic features similar to those obtained from real data. For this purpose, we calculate the empirical expectation of the 900-dimensional sentence feature vector over 2,000 real sentences and 2,000 synthetic sentences. As shown in Figure 3(left), the expectation of these 900 feature dimensions from synthetic sentences matches well with the feature expectation from the real sentences. We also compared the estimated covariance matrix elements $\tilde{\mathbf{\Sigma}}_{i,j,\boldsymbol{f}}$ (including $900 * 899/2$ off-diagonal elements and 900 diagonal elements) from real data against the covariance matrix elements $\tilde{\mathbf{\Sigma}}_{i,j,\tilde{\boldsymbol{f}}}$ estimated from synthetic data, in Figure 3(right). We observe that the covariance structure of the 900-dimensional features from real and synthetic sentences in general match well. The full covariance matrices for real and synthetic sentences are provided in

Table 1. Quantitative results using BLEU-2,3,4 and KDE.

	BLEU-4	BLEU-3	BLEU-2	KDE(nats)
AE	0.01±0.01	0.11±0.02	0.39±0.02	2727±42
VAE	0.02±0.02	0.16±0.03	0.54±0.03	1892±25
seqGAN	0.04±0.04	0.30±0.08	0.67±0.04	2019±53
textGAN(MM)	0.09±0.04	0.42±0.04	0.77±0.03	1823±50
textGAN(CM)	0.12±0.03	0.49±0.06	0.84±0.02	1686±41
textGAN(MMD)	**0.13±0.05**	0.49±0.06	0.83±0.04	1688±38
textGAN(MMD-L)	0.11±0.05	**0.52±0.07**	**0.85±0.04**	**1684±44**

the Supplementary Material. We observe that the (mapped) synthetic features nicely cover the real sentence features density, while "completing" other areas of low density.

Quantitative comparison We evaluate the generated-sentence quality using the BLEU score (Papineni et al., 2002) and Kernel Density Estimation (KDE), as in Goodfellow et al. (2014); Nowozin et al. (2016). For comparison, we consider textGAN with 4 different loss objectives: Mean Matching (MM) as in Salimans et al. (2016), Covariance Matching (CM) as in (5), MMD and MMD with compressed network (MMD-L), by mapping the original 900-dimensional features to 200-dimensional, as described in Section 2.3. We also compare to a baseline autoencoder (AE) model. The AE uses a CNN as encoder and an LSTM as decoder, where the CNN and LSTM network structures are set to be identical as the CNN and LSTM used in textGAN. We finally consider a Variational Autoencoder (VAE) implemented as in Bowman et al. (2016). To train the VAE model, we use annealing to gradually increase the KL divergence between the prior and approximated posterior. The details are provided in the the Supplementary Material. We also compare with seqGAN (Yu et al., 2017). For seqGAN we follow the authors' guidelines of running 350 pre-training epochs followed by 50 discriminator training epochs, to generate 320 sentences. For AE, VAE and textGAN, we first uniformly sample 320 latent codes from the latent code space, and use the corresponding generator (or decoder, in the AE/VAE case) to generate sentences.

For BLEU score evaluation, we follow the strategy in Yu et al. (2017) of using the entire test set as the reference. For KDE evaluation, the lengths of the generated sentences are different, thus we first embed all the sentences to a 900-dimensional vector. Since no standard sentence encoder is available, we use the encoder learned from AE. The covariance matrix for the Parzen kernel in KDE is set to be the covariance of feature vectors for real tested sentences. Despite the fact that the KDE approach, as a log-likelihood estimator tends to have high variance (Theis et al., 2016), the KDE score tracks well with our BLEU score evaluation.

The results are shown in Table 1. MMD and MMD-L generally score higher in sentences quality. MMD-L seems better at capturing 2-grams (BLEU-2), while MMD outperforms MMD-L in 4-grams (BLEU-4). We also observed that when using CM, the generated sentences tend to be shorter than

Table 2. Sentences generated by textGAN.

a	we show the joint likelihood estimator (in a large number of estimating variables embedded on the subspace learning) .
b	this problem achieves less interesting choices of convergence guarantees on turing machine learning .
c	in hidden markov relational spaces , the random walk feature decomposition is unique generalized parametric mappings.
d	i see those primitives specifying a deterministic probabilistic machine learning algorithm .
e	i wanted in alone in a gene expression dataset which do n't form phantom action values .
f	as opposite to a set of fuzzy modelling algorithm , pruning is performed using a template representing network structures .

MMD (not shown).

Generated sentences Table 2 shows six sentences generated by textGAN. Note that the generated sentences seem to be able to produce novel phrases by imagining concept combinations, *e.g.*, in Table 2(b,c,f), or to borrow words from a different corpus to compose novel sentences, *e.g.*, in Table 2(d,e). In many cases, it learns to automatically match the parentheses and quotation marks, *e.g.*, in Table 2(a), and can synthesize relatively long sentences, *e.g.*, in 2(a,f). In general, the synthetic sentences seem syntactically reasonable. However, the semantic meaning is less well preserved especially in sentence of more than 20 words, *e.g.*, in Table 2(e,f).

We observe that the discriminator can still sufficiently distinguish the synthetic sentences from the real ones (the probability to predict synthetic data as real is around 0.05), even when the synthetic sentences seems to perserve reasonable grammatical structure and use proper wording. It is likely that the CNN is able to accurately characterize the semantic meaning and differentiate sentences, while the generator may get trapped into a local optimum, where any slight modification would result in a higher loss (3) for the generator. Presumably, long-range distance features are not difficult to abstract by the discriminator/encoder, however, is less likely to be imitated by the generator. One promising direction is to leverage reinforcement learning strategies as in Yu et al. (2017), where the updating for LSTM can be more effectively steered. Nevertheless, investigation on how to improve the the long-range behavior is left as interesting future work.

Latent feature space trajectories Following Bowman et al. (2016), we further empirically evaluate whether the latent variable space can "densely" encode sentences. We visualize the transition from one sentence to another by constructing a linear path between two randomly selected points in latent feature space, to then generate the intermediate sentences along the linear trajectory. For comparison, a baseline autoencoder (AE) is trained for 20 epochs. The results for textGAN and AE are presented in Table 3. Compared to AE, the sentences produced by textGAN are generally more syntactically and semantically reasonable. The transition suggest "smoothness" and interpretability, however, the wording choices and sentence structure showed dramatic changes in some regions in the latent feature space. This seems to indicate that local "transition smoothness" varies from region to region.

Table 3. Intermediate sentences produced from linear transition between two points (A and B) in the latent feature space. Each sentence is generated from a latent point on a linear path.

	textGAN	**AE**
A	our methods apply novel approaches to solve modeling tasks .	
-	our methods apply novel approaches to solve modeling .	our methods apply to train UNK models involving complex .
-	our methods apply two different approaches to solve computing .	our methods solve use to train) .
-	our methods achieves some different approaches to solve computing .	our approach show UNK to models exist .
-	our methods achieves the best expert structure detection .	that supervised algorithms show to UNK speed .
-	the methods have been different related tasks .	that address algorithms to handle) .
-	the guy is the minimum of UNK .	that address versions to be used in .
-	the guy is n't easy tonight .	i believe the means of this attempt to cope .
-	i believe the guy is n't smart okay?	i believe it 's we be used to get .
-	i believe the guy is n't smart .	i believe it i 'm a way to belong .
B	i believe i 'm going to get out .	

5. Conclusion

We have introduced a novel approach for text generation using adversarial training, termed TextGAN, and have discussed several techniques to specify and train such a model. We demonstrated that the proposed model delivers superior performance compared to related approaches, can produce realistic sentences, and that the learned latent representation space can "smoothly" encode plausible sentences. We quantitatively evaluate the proposed methods with baseline models and existing methods. The results indicate superior performance of TextGAN.

In future work, we will attempt to apply conditional GAN models (Mirza & Osindero, 2014) to disentangle the latent representations for different writing styles. This would enable a smooth lexical and grammatical transition between different writing styles. It would be also interesting to generate text by conditioning on observed images (Pu et al., 2016). In addition, we plan to leverage an additional refining stage where a reverse-order LSTM (Graves & Schmidhuber, 2005) is applied after the sentence is first generated, to produce sentences with better long-term semantical interpretation.

Acknowledgments

This research was supported by ARO, DARPA, DOE, NGA, ONR and NSF.

References

Arjovsky, Martin and Bottou, Léon. Towards principled methods for training generative adversarial networks. In *ICLR*, 2017.

Arjovsky, Martin, Chintala, Soumith, and Bottou, Léon. Wasserstein gan. In *ICML*, 2017.

Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I., Bergeron, A., Bouchard, N., Warde-Farley, D., and Bengio, Y. Theano: new features and speed improvements. *arXiv:1211.5590*, 2012.

Bengio, Samy, Vinyals, Oriol, Jaitly, Navdeep, and Shazeer, Noam. Scheduled sampling for sequence prediction with recurrent neural networks. In *NIPS*, 2015.

Bowman, Samuel R, Vilnis, Luke, Vinyals, Oriol, Dai, Andrew M, Jozefowicz, Rafal, and Bengio, Samy. Generating sentences from a continuous space. In *CoNLL*, 2016.

Chen, Xi, Duan, Yan, Houthooft, Rein, Schulman, John, Sutskever, Ilya, and Abbeel, Pieter. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *NIPS*, 2016.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *EMNLP*, 2014.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. Natural language processing (almost) from scratch. In *JMLR*, 2011.

Donahue, Jeff, Krähenbühl, Philipp, and Darrell, Trevor. Adversarial feature learning. In *ICLR*, 2017.

Dziugaite, Gintare Karolina, Roy, Daniel M, and Ghahramani, Zoubin. Training generative neural networks via maximum mean discrepancy optimization. *arXiv:1505.03906*, 2015.

Gan, Zhe, Pu, Yunchen, Henao, Ricardo, Li, Chunyuan, He, Xiaodong, and Carin, Lawrence. Unsupervised learning of sentence representations using convolutional neural networks. *arXiv preprint arXiv:1611.07897*, 2016.

Goodfellow, Ian, Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, and Bengio, Yoshua. Generative adversarial nets. In *NIPS*, 2014.

Graves, Alex and Schmidhuber, Jürgen. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 2005.

Gretton, Arthur, Borgwardt, Karsten M, Rasch, Malte J, Schölkopf, Bernhard, and Smola, Alexander. A kernel two-sample test. *JMLR*, 2012.

Gumbel, Emil Julius and Lieblein, Julius. Statistical theory of extreme values and some practical applications: a series of lectures. 1954.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. In *Neural computation*, 1997.

Hu, B., Lu, Z., Li, H., and Chen, Q. Convolutional neural network architectures for matching natural language sentences. In *NIPS*, 2014.

Huszár, Ferenc. How (not) to train your generative model: Scheduled sampling, likelihood, adversary? *arXiv:1511.05101*, 2015.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

Jang, Eric, Gu, Shixiang, and Poole, Ben. Categorical reparameterization with gumbel-softmax. In *ICLR*, 2017.

Johnson, R. and Zhang, T. Effective use of word order for text categorization with convolutional neural networks. In *NAACL HLT*, 2015.

Kalchbrenner, N., Grefenstette, E., and Blunsom, P. A convolutional neural network for modelling sentences. In *ACL*, 2014.

Kim, Y. Convolutional neural networks for sentence classification. In *EMNLP*, 2014.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Kingma, Diederik P and Welling, Max. Auto-encoding variational bayes. In *ICLR*, 2014.

Lamb, Alex M, GOYAL, Anirudh Goyal ALIAS PARTH, Zhang, Ying, Zhang, Saizheng, Courville, Aaron C, and Bengio, Yoshua. Professor forcing: A new algorithm for training recurrent networks. In *NIPS*, 2016.

Larsen, Anders Boesen Lindbo, Sønderby, Søren Kaae, Larochelle, Hugo, and Winther, Ole. Autoencoding beyond pixels using a learned similarity metric. In *ICML*, 2016.

Li, Jiwei, Monroe, Will, Ritter, Alan, Galley, Michel, Gao, Jianfeng, and Jurafsky, Dan. Deep reinforcement learning for dialogue generation. In *EMNLP*, 2016.

Li, Jiwei, Monroe, Will, Shi, Tianlin, Ritter, Alan, and Jurafsky, Dan. Adversarial learning for neural dialogue generation. *arXiv:1701.06547*, 2017.

Li, Yujia, Swersky, Kevin, and Zemel, Richard S. Generative moment matching networks. In *ICML*, 2015.

Liu, Qiang, Lee, Jason D, and Jordan, Michael I. A kernelized stein discrepancy for goodness-of-fit tests. In *ICML*, 2016.

Maaten, Laurens van der and Hinton, Geoffrey. Visualizing data using t-sne. *JMLR*, 2008.

Maddison, Chris J, Mnih, Andriy, and Teh, Yee Whye. The concrete distribution: A continuous relaxation of discrete random variables. In *ICLR*, 2017.

Makhzani, Alireza, Shlens, Jonathon, Jaitly, Navdeep, Goodfellow, Ian, and Frey, Brendan. Adversarial autoencoders. *arXiv:1511.05644*, 2015.

Mescheder, Lars, Nowozin, Sebastian, and Geiger, Andreas. Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks. In *ICML*, 2017.

Metz, Luke, Poole, Ben, Pfau, David, and Sohl-Dickstein, Jascha. Unrolled generative adversarial networks. In *ICLR*, 2017.

Micchelli, Charles A, Xu, Yuesheng, and Zhang, Haizhang. Universal kernels. *JMLR*, 2006.

Mirza, Mehdi and Osindero, Simon. Conditional generative adversarial nets. *arXiv:1411.1784*, 2014.

Nowozin, Sebastian, Cseke, Botond, and Tomioka, Ryota. f-gan: Training generative neural samplers using variational divergence minimization. In *NIPS*, 2016.

Papineni, Kishore, Roukos, Salim, Ward, Todd, and Zhu, Wei-Jing. Bleu: a method for automatic evaluation of machine translation. In *ACL*, 2002.

Pu, Yunchen, Gan, Zhe, Henao, Ricardo, Yuan, Xin, Li, Chunyuan, Stevens, Andrew, and Carin, Lawrence. Variational autoencoder for deep learning of images, labels and captions. In *NIPS*, 2016.

Ramdas, Aaditya, Reddi, Sashank J, Poczos, Barnabas, Singh, Aarti, and Wasserman, Larry. On the high-dimensional power of linear-time kernel two-sample testing under mean-difference alternatives. *arXiv:1411.6314*, 2014.

Salimans, Tim, Goodfellow, Ian, Zaremba, Wojciech, Cheung, Vicki, Radford, Alec, and Chen, Xi. Improved techniques for training gans. In *NIPS*, 2016.

Sutskever, I., Vinyals, O., and Le, Q. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

Theis, Lucas, Oord, Aäron van den, and Bethge, Matthias. A note on the evaluation of generative models. In *ICLR*, 2016.

Wang, Dilin and Liu, Qiang. Learning to draw samples: With application to amortized mle for generative adversarial learning. *arXiv:1611.01722*, 2016.

Williams, Ronald J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 1992.

Yu, Lantao, Zhang, Weinan, Wang, Jun, and Yu, Yong. Seqgan: sequence generative adversarial nets with policy gradient. In *AAAI*, 2017.

Zhang, Yizhe, Gan, Zhe, and Carin, Lawrence. Generating text via adversarial training. In *NIPS Workshop on Adversarial Training*, 2016.

Zhao, Junbo, Mathieu, Michael, and LeCun, Yann. Energy-based generative adversarial network. In *ICLR*, 2017.

Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., and Fidler, S. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *ICCV*, 2015.

A. Additional results

A.1. RNN specifications

We leverage a LSTM RNN for our generator. Each LSTM unit has a cell containing a state $\boldsymbol{c}_t$ at time t. Reading or writing the memory unit is controlled through sigmoid gates, namely, input gate $\boldsymbol{i}_t$, forget gate $\boldsymbol{g}_t$, and output gate $\boldsymbol{o}_t$. The hidden units $\boldsymbol{h}_t$ are updated as follows:

$$\boldsymbol{i}_t = \sigma(\mathbf{W}_i \boldsymbol{y}_{t-1} + \mathbf{U}_i \boldsymbol{h}_{t-1} + \mathbf{C}_i \boldsymbol{z}) \tag{9}$$
$$\boldsymbol{g}_t = \sigma(\mathbf{W}_g \boldsymbol{y}_{t-1} + \mathbf{U}_g \boldsymbol{h}_{t-1} + \mathbf{C}_g \boldsymbol{z}) \tag{10}$$
$$\boldsymbol{o}_t = \sigma(\mathbf{W}_o \boldsymbol{y}_{t-1} + \mathbf{U}_o \boldsymbol{h}_{t-1} + \mathbf{C}_o \boldsymbol{z}) \tag{11}$$
$$\tilde{\boldsymbol{c}}_t = \tanh(\mathbf{W}_c \boldsymbol{y}_{t-1} + \mathbf{U}_c \boldsymbol{h}_{t-1} + \mathbf{C}_c \boldsymbol{z}) \tag{12}$$
$$\boldsymbol{c}_t = \boldsymbol{g}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t \tag{13}$$
$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t)\,, \tag{14}$$

where $\sigma(\cdot)$ denotes the logistic sigmoid function, and $\odot$ represents the element-wise multiply operator (Hadamard product). $\mathbf{W}_{\{i,g,o,c\}}, \mathbf{U}_{\{i,g,o,c\}}$, $\mathbf{C}_{\{i,g,o,c\}}$, $\mathbf{V}$ and $\mathbf{C}$ are the set of parameters. Note that $\boldsymbol{z}$ is used as an explicit input at each time step of the LSTM to guide the generation of $\tilde{s}$. Another remark is that all the randomness in the generator come from z. The synthetic sentence $\tilde{s}$ is deterministically obtained given z.

A.2. Universality of the embedded kernel

In below we consider the universality of a kernel defined on the input space $\mathcal{X}$, constructed by a universal kernel on feature space $\mathcal{F}$.

Proposition 1. *Suppose a continuous universal kernel $k : \mathcal{F} \times \mathcal{F} \to \mathbb{R}$ is a universal kernel. If a space $\mathcal{X}$ has a continuous bijective mapping $\lambda : \mathcal{X} \to \mathcal{F}$, the composed kernel $\tilde{k} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, such that for $\forall x, x' \in \mathcal{X}$, $\tilde{k}(x, x') = k(f(x), f(x'))$ is a universal kernel define on $\mathcal{X}$.*

Proof. Denote $\mathbb{N}_n = \{1, 2, \cdots, n\}$. Since k is a universal kernel, from Micchelli et al. (2006), k is continuous and its RKHS $\mathcal{H}$ is dense in $C(\mathcal{F}) \triangleq \{g : \mathcal{F} \to \mathbb{R} | g\text{continuous}\}$, *i.e.*, for any n points $\{f_i\}_{i \in \mathbb{N}_n} \in \mathcal{F}$, for $\forall g \in C(\mathcal{F})$, there exists $a_i : i \in \mathbb{N}_n$ that

$$g(\cdot) = \sum_{i \in \mathbb{N}_n} a_i k(f_i, \cdot)\,, \tag{15}$$

where $k(f_i, \cdot) \in \mathcal{H}$. This is known as the *universal approximation* property. Since λ is a bijective function, consider any $\{x_i\}_{i \in \mathbb{N}_n} \in \mathcal{X}$. By construction, for $\forall h \in C(\mathcal{X}) \triangleq h : \mathcal{X} \to \mathbb{R} | h\text{continuous}$, consider $g = h \circ \lambda^{-1}$, from (15) we have

$$h(x) = h(\lambda^{-1}(\lambda(x))) = g(\lambda(x)) \tag{16}$$
$$= \sum_{i \in \mathbb{N}_n} a_i k(\lambda(x_i), \cdot) = \sum_{i \in \mathbb{N}_n} a_i \tilde{k}(x_i, \cdot)\,. \tag{17}$$

Hence the $\tilde{k}$ is a universal kernel. □

A.3. Alternative upper bound objective

In this subsection we show that (5) corresponds to an upper bound of the JSD between two Gaussian distribution.

Proof. The KL divergence $D_{KL}(p||q)$ between two multivariate Gaussian distribution $p(x) \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and $q(x) \sim \mathcal{N}(\tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\Sigma}})$ is given by

$$D_{KL}(p||q) = \frac{1}{2}\Big[\text{tr}(\tilde{\boldsymbol{\Sigma}}^{-1}\boldsymbol{\Sigma}) + \log\frac{|\tilde{\boldsymbol{\Sigma}}|}{\boldsymbol{\Sigma}} + \tag{18}$$
$$+ (\tilde{\boldsymbol{\mu}} - \boldsymbol{\mu})^T(\tilde{\boldsymbol{\Sigma}}^{-1})(\tilde{\boldsymbol{\mu}} - \boldsymbol{\mu}) - d\Big]\,. \tag{19}$$

We first start with a proposition.

Proposition 2. *Assume three arbitrary valid continuous density functions p, q, r has probability measures over a domain $\mathcal{X}$, we have that $D_{KL}(p||q) + D_{KL}(p||r) \geq D_{KL}(p||(q + r)/2)$.*

The proof is as follows.

$$D_{KL}(p||q) + D_{KL}(p||r) \tag{20}$$
$$= \int -p(x)\log\frac{q(x)}{p(x)} + \log\frac{r(x)}{p(x)} \tag{21}$$
$$<= \int -p(x)\log\frac{[q(x) + r(x)]/2}{p(x)} \tag{22}$$
$$= D_{KL}(p||(q + r)/2)\,. \tag{23}$$

From (19) and (23), following the definition of JSD, we have

$$JSD(p||q) \tag{24}$$
$$= D_{KL}\Big(p\Big|\Big|\frac{p+q}{2}\Big) + D_{KL}\Big(q\Big|\Big|\frac{p+q}{2}\Big) \tag{25}$$
$$<= D_{KL}(p||p) + D_{KL}(p||q) \tag{26}$$
$$+ D_{KL}(q||q) + D_{KL}(q||p) \tag{27}$$
$$= \frac{1}{2}\Big[\text{tr}(\tilde{\boldsymbol{\Sigma}}^{-1}\boldsymbol{\Sigma} + \boldsymbol{\Sigma}^{-1}\tilde{\boldsymbol{\Sigma}}) + (\tilde{\boldsymbol{\mu}} - \boldsymbol{\mu})^T(\tilde{\boldsymbol{\Sigma}}^{-1} + \boldsymbol{\Sigma}^{-1})(\tilde{\boldsymbol{\mu}} - \boldsymbol{\mu}) - 2d\Big]\,. \tag{28}$$

Therefore, (5) is an upper bound for $JSD(p||q)$. Directly minimizing $JSD(p||q)$ is hard, however (5) is more tractable. □

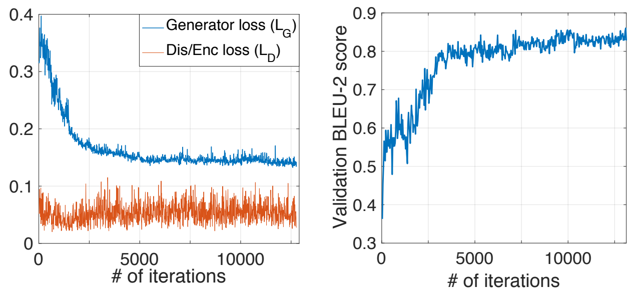

Figure 4. Left: learning curve for TextGAN. Right: validation BLEU-2 score.

A.4. VAE and KL divergence annealing

In VAE we optimize the lower bound objective below.

$$\mathcal{L} = \mathbb{E}_{q(z|x)} \log p(x|z) + \alpha \mathbb{E}_{q(z|x)} [\log p(z) - \log q(z|x)] \, ,$$

where α is a scaling parameter. We observed that directly minimizing the above objective with $\alpha = 1$ would fail to converge. Thus we train the VAE under an annealing scheme, where α_t is set to be $\min(t/50,000, 1)$ and t is the number of iterations that has been performed.

A.5. Feature matching results

The covariance matrices of real features $\boldsymbol{f}$ and synthetic features $\tilde{\boldsymbol{f}}$ are shown Figure 5. Each of the covariance matrices is computed over 2,000 data points. Learning curves are shown in Figure 4.

A.6. Experimental setup

To accelerate convergence, we begin each run with a warm-up training. Specifically, This warm-up includes: (*i*) using a mean matching objective for the generator loss, *i.e.*, $||\mathbb{E}\boldsymbol{f} - \mathbb{E}\tilde{\boldsymbol{f}}||^2$, as in Salimans et al. (2016); (*ii*) trimming the generated sentences if the length exceeds 15, by removing words afterwards. For MMD Gaussian kernel, we set the bandwidth parameters as $[10, 15, 20, 25, 30]$ in our experiments. The L used in soft-argmax is $10,000$. The λ_r and λ_m are set to be 0.01 and 0.001. The activation function employed in discriminator/encoder is hyperbolic tangent function. We also utilized several other training techniques in order to stabilize training, including soft-labeling (Salimans et al., 2016) and batch normalization (Ioffe & Szegedy, 2015). For soft-labeling the discriminator is constraint to maximally assign 0.99 and minimally assign 0.01 for the probability of being from real data. The batch normalization is added on the CNN output before activation function. In practice we find batchnorm does not provide significant performance benefits.

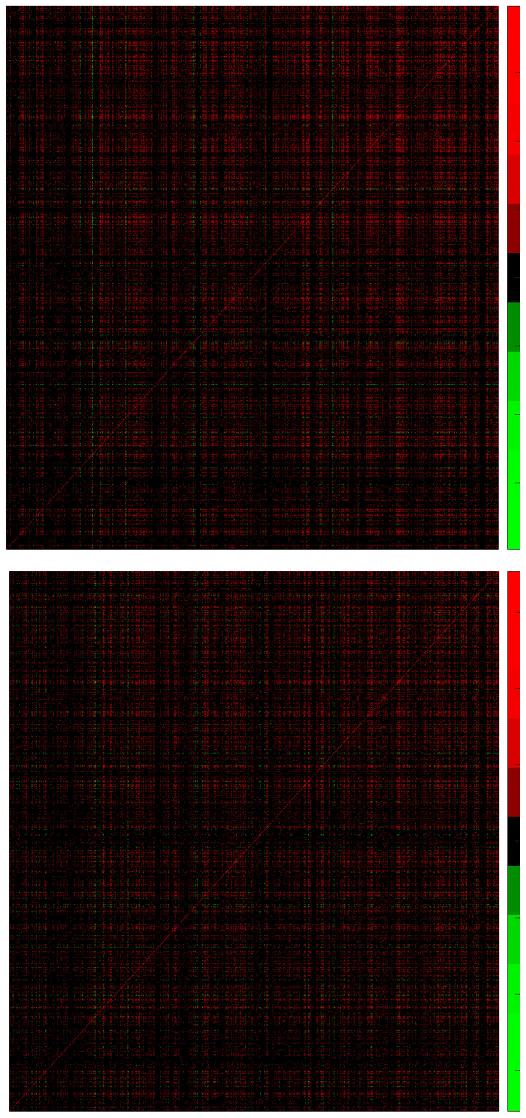

Figure 5. Covariance matching. Upper: synthetic features covariance. Lower: real features covariance.